

A UNIFIED FORMAL DESCRIPTION OF BEHAVIOURAL AND DELIBERATIVE ROBOTIC MULTI-AGENT SYSTEMS

Cezary Zieliński^{*,1}

^{*} *Institute of Control and Computation Engineering,
 Warsaw University of Technology, ul. Nowowiejska 15/19,
 00-665 Warsaw, POLAND*

Abstract: Many behavioural (Arkin, 1998) and deliberative (Russell and Norvig, 1995) robot control methods have been developed. Besides investigating task execution by a single robot multi-agent systems are being developed. Unfortunately only some of those control methods have a formal description. To aggravate the situation, those control methods that have formal specifications use different formal means of description. This paper presents an outline of a unified formal description of systems composed of many embodied agents using behavioural or deliberative control methods. Such a description is needed to specify in an implementation independent way the functionality of programming frameworks (Markiewicz and Lucena, 2001). Transition function based approach to the formal description of the above mentioned systems has been assumed to facilitate the future implementation of thus specified programming frameworks. The presented description is a generalization of the formal description used for the specification of MRROC++ (Zieliński, 1999; Zieliński, 2001). Copyright ©2003 IFAC

Keywords: Multi-agent systems, robot programming

1. INTRODUCTION

This paper uses the definition of an agent as formulated in (Russell and Norvig, 1995): *An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors.* The paper concentrates on agents acting in physical environments, and not on abstract agents, e.g. acting in the World Wide Web. Agents operating in a physical environment gather information about its state through receptors (i.e. hardware sensors) and influence its state through their effectors (e.g., actuators or motors combined with mechanical limbs, manipulator arm, leg or legs, wheels). Such arti-

facts are called **embodied** or **robotic agents**. The paper concentrates on the design of the software component of systems composed of such agents, and especially the formal tools for designing this software, i.e. programming frameworks.

In general, **programming frameworks** are application generators for a specific domain of problems (Markiewicz and Lucena, 2001). This paper concentrates on robot programming frameworks, i.e. libraries of software modules (procedures, objects or processes) supplemented by design patterns for robot system controllers. A design pattern is composed of the so called frozen spots and hot spots. **Frozen spots** are the unalterable parts of the generated software and **hot spots** are the variable parts. Code of hot spots is delivered by the system builder or extracted from

¹ Partially supported by Warsaw University of Technology grant no. 503G/1031/0003/002

the library of ready modules. Although the terms: *framework* and *hot or frozen spots* are fairly recent, the concept of providing a library of ready or modifiable modules that can be assembled according to a certain pattern into a ready to use application (e.g. controller), is quite old. Formerly frameworks used to be called simply robot programming or control libraries or languages, but both of those terms are not adequate. A library does not imply an associated software pattern into which the modules should be inserted, and a language is usually associated with a specific grammar (e.g. syntactic structure). As general purpose languages, such as Pascal or C have been used as the development tools for those libraries, so no new language was being defined. Although initially specialised programming languages had been favored, they lost they appeal, when it turned out that in the robotics domain the variability of equipment causes an ever greater demand for extensibility of those languages. Any extensions force a modification of the compiler or the interpreter of the language rendering the alteration more costly. Moreover, soon it became obvious that the specialised robot programming languages have to provide nearly all the capabilities of a general purpose programming language. Under those circumstances it was more reasonable to use a general purpose programming language and a library of modules specific to robot control and to define a general pattern according to which they should be assembled. Thus, although specialised robot programming languages gained considerable popularity (e.g. WAVE (Paul, 1977), AL (Mujtaba and Goldman, 1979; Zieliński, 1992), VAL II (*User's Guide to VAL II: Programming Manual Ver.2.0*, 1986), AML (Taylor *et al.*, 1982), RAPT (Ambler and Corner, 1984; Zieliński, 1992), SRL (Blume and Jakob, 1984; Blume and Jakob, 1986), TORBOL (Zieliński, 1991; Zieliński, 1992)) robot programming frameworks have been especially favored by the research community (e.g. RCCL (Hayward and Paul, 1986), Multi-RCCL (Lloyd and Hayward, 1993), KALI (Hayward and Hayati, 1988; Hayward *et al.*, 1989; Backes *et al.*, 1989), PASRO (Blume and Jakob, 1985; Blume and Jakob, 1986), RORC (Zieliński, 1995b; Zieliński, 1993), MRROC (Zieliński, 1995b; Zieliński, 1995a), MRROC++ (Zieliński, 1997; Zieliński, 1999), GENOM (Fleury and Herrb, 2001; Alami *et al.*, 1998), DCA (Pettersson *et al.*, 2001), TCA (Simmons *et al.*, 1997), TDL (Simmons and Apfelbaum, 1998), Generis (Morales, 1999), SmartSoft (Schlegel and Wörz, 1999)). This paper presents a tool for formal specification of programming frameworks for systems composed of heterogeneous embodied agents. This description takes into account that:

- It has to facilitate both the specification of the software of the designed system and its

implementation. Nevertheless, the means of the implementation (i.e. programming language, operating system *etc.*) should not be predetermined *a priori*.

- It should be flexible enough to encompass diverse structures of both the system and its components.
- The specification of behavioural agents, deliberative agents or heterogeneous agents should be possible.
- The number and type of both the effectors and the sensors used should not be fixed.

Summarizing, the paper proposes a formal method of describing multi-agent systems lending itself to structuring and implementation of a general programming framework for producing such systems. The presented formalism is a generalization of the approach used to define and implement the MRROC++ (Zieliński *et al.*, 1998; Zieliński, 1999; Zieliński, 2001; Zieliński *et al.*, 2001) robot programming framework. MRROC++ based systems have either a hierarchical structure or a structure composed of independent entities. In both cases no direct interaction between the agents is possible. Obviously indirect interactions through sensing or upper layers of hierarchy are possible. The presented formalism includes direct interactions between agents.

2. EMBODIED AGENTS

For the purpose of brevity, and because of contextual obviousness, the denotations assigned to the components of the considered system and their state will not be distinguished.

Let us consider a multi-agent system consisting of n_a agents. The state of a single embodied agent a_j , $j = 1, \dots, n_a$, is:

$$s_j = \langle c_j, e_j, V_j \rangle \quad (1)$$

where:

- c_j – state of the **control subsystem** of the agent (i.e. memory: variables, program *etc.*),
- e_j – state of the **effector** of the agent,
- V_j – bundle of **virtual sensor** readings utilised by the agent.

A bundle of virtual sensor readings contains n_v individual virtual sensor readings:

$$V_j = \langle v_{j1}, \dots, v_{jn_v} \rangle \quad (2)$$

Each virtual sensor v_{jk} , $k = 1, \dots, n_v$, produces an aggregate reading from one or more exteroceptors. Exteroceptors, for brevity further on called simply **receptors** or real sensors, include all the measuring devices gathering information from the

environment of the system. Interoceptors and proprioceptors, i.e. devices for measuring the internal state of the system (e.g. position encoders, resolvers), are not treated here as receptors, because they supply data about the state of the effectors and not the environment of the agent, and thus are associated with the effector. The data obtained from the receptors usually cannot be used directly in agent motion control, e.g. to control the arm motion, one would need the grasping location of the object that is to be picked and not the whole bit-map delivered by a camera. In some other cases a simple sensor in its own right would not suffice to control the motion (e.g. a single touch sensor), but several such sensors deliver meaningful data. The process of extracting meaningful information for the purpose of motion control is named data aggregation and is performed by virtual sensors. Thus the k th virtual sensor reading obtained by the agent a_j is formed as:

$$v_{jk} = f_{v_{jk}}(c_j, R_{jk}) \quad (3)$$

where R_{jk} is a bundle of receptor readings used for the creation of the k th virtual sensor reading.

$$R_{jk} = \langle r_{j_{k1}}, \dots, r_{j_{kn_r}} \rangle \quad (4)$$

where n_r is the number of receptor readings $r_{j_{kl}}$, $l = 1 \dots n_r$ taken into account in the process of forming the reading of the k th virtual sensor of the agent a_j . It should be noted that (3) implies that the reading of the virtual sensor depends also on c_j . In this way the agent has the capability of configuring the sensor as well as delivering to the virtual sensor the relevant information about the current state of the agent (including its effector). This might be necessary in the case of computing the reading of a virtual sensor having its associated receptors mounted on the effector (e.g. artificial skin).

The responsibility of the agent's control subsystem c_j is to: gather information about the state of the environment through the associated virtual sensor bundle V_j , obtain the information from the other agents $a_{j'}$ ($j' \neq j$), monitor the state of its own effector e_j , and to process all of this information to produce: a new state of the effector e_j , influence the future functioning of the virtual sensors V_j , and send out information to the other agents $a_{j'}$. As a side effect the internal state of the control subsystem c_j changes. To do this effectively several components of the control subsystem can be distinguished:

- c_{e_j} – image of the effector (this is a perception of the effector by the control subsystem, e.g. motor shaft positions, joint angles, end-effector location),

- c_{V_j} – images of the virtual sensors (i.e. current virtual sensor reading and configuration),
- c_{T_j} – inter-agent transmission (i.e. information mutually transmitted between the agents),
- c_{c_j} – all the other relevant variables.

Thus the the control subsystem is partitioned:

$$c_j = \langle c_{c_j}, c_{e_j}, c_{V_j}, c_{T_j} \rangle \quad (5)$$

From the point of view of the system designer the state of the control subsystem changes at a servo sampling rate or a low multiple of that. If i denotes the current instant, the next considered instant is denoted by $i+1$. The control subsystem uses $c_j^i = \langle c_{c_j}^i, c_{e_j}^i, c_{V_j}^i, c_{T_j}^i \rangle$ to produce $c_j^{i+1} = \langle c_{c_j}^{i+1}, c_{e_j}^{i+1}, c_{V_j}^{i+1}, c_{T_j}^{i+1} \rangle$, i.e.:

$$\begin{cases} c_{c_j}^{i+1} = f_{c_{c_j}}(c_{c_j}^i, c_{e_j}^i, c_{V_j}^i, c_{T_j}^i) \\ c_{e_j}^{i+1} = f_{c_{e_j}}(c_{c_j}^i, c_{e_j}^i, c_{V_j}^i, c_{T_j}^i) \\ c_{V_j}^{i+1} = f_{c_{V_j}}(c_{c_j}^i, c_{e_j}^i, c_{V_j}^i, c_{T_j}^i) \\ c_{T_j}^{i+1} = f_{c_{T_j}}(c_{c_j}^i, c_{e_j}^i, c_{V_j}^i, c_{T_j}^i) \end{cases} \quad (6)$$

or more compactly:

$$c_j^{i+1} = f_{c_j}(c_j^i) \quad (7)$$

The control subsystem obtains the input values $c_{e_j}^i, c_{V_j}^i, c_{T_j}^i$ through transmission from the other components of the agent itself (e.g. effector, virtual sensors) or the other agents. It also must transmit the computed values $c_{e_j}^{i+1}, c_{V_j}^{i+1}, c_{T_j}^{i+1}$ to the other elements of the agent or the other partners. If we do not want to make any assumptions about the order of those transmissions and do not assume that the input and output images are of the same type (this occurs very rarely) duplicates of those entities must be stored. Only the internal variables c_{c_j} should have a single representation. Hence the control subsystem state can be represented as (fig. 1):

$$c_j = \langle c_{c_j}, c_{e_j}^i, c_{e_j}^{i+1}, c_{V_j}^i, c_{V_j}^{i+1}, c_{T_j}^i, c_{T_j}^{i+1} \rangle \quad (8)$$

3. A MULTI-AGENT SYSTEM

The needs of communication between the agents are defined by the arguments of the transition functions (6). If the computation of any of the output values requires some input data, this input data has to be delivered to the agent's control subsystem, and this defines a connection between the source of this data and the agent's control subsystem.

The agents of the described system can act: purely independently, they can interact directly through

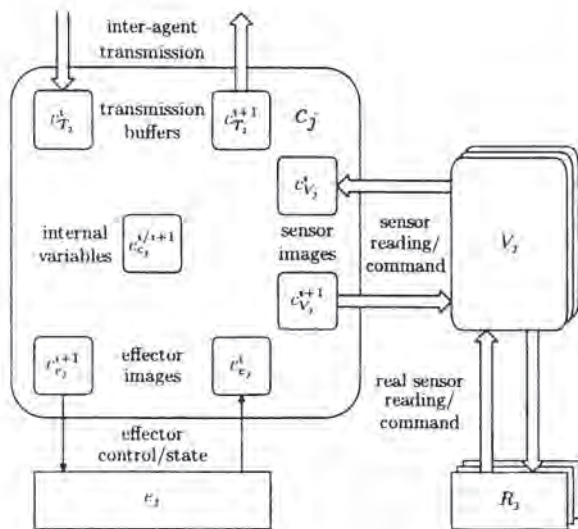


Fig. 1. A single embodied agent a_j , $j = 1, \dots, n_a$ an exchange of transmitted data (c_{T_j}), they can also interact indirectly by sensing (through the receptors) the other agents or the results of their actions, or they can be coordinated by a hierarchically higher entity, i.e. a coordinator. The **coordinator**, for the sake of consistency, can be treated as an abstract agent, i.e. an agent that does not possess an effector (a body). Nevertheless, there is no reason to assume that the abstract agent should not gather directly the information from the environment, thus it can have a virtual sensor bundle of its own. In this case it would deliver global information about the environment, e.g. a camera gathering the information about the global state of the football pitch in a RoboCup match. Let us distinguish the coordinator by the subscript 0. In the face of the above its state will be:

$$s_0 = \langle c_0, V_0 \rangle \quad (9)$$

Hence the system state is described by:

$$s = \langle s_0, s_1, \dots, s_{n_e} \rangle \quad (10)$$

The structure of the coordinator will not be discussed further in this paper. Let it suffice to say that it can be subdivided into other abstract sub-agents.

4. INTERNAL FUNCTIONING OF AN AGENT

Internal functioning of an agent is defined by the transition functions (6) or (7). The flexibility of a programming framework is located in the ability of expressing diverse approaches to programming the actions of each agent, and so the formal description should enable easy formulation of diverse control strategies. Usually instead of computing a single set of functions (6) many partial functions

are evaluated and the final result is obtained by a certain composition of the partial results. Thus n_f partial functions are defined:

$$c_j^{i+1} = m f_{c_j}(c_j^i), \quad m = 1, \dots, n_f \quad (11)$$

Moreover, the partial functions do not have to take as arguments all of the components of c_j^i . The following subsections investigate some of the interesting possibilities.

4.1 Purely reactive systems

In the case of a purely reactive system the choice of the function $m f_{c_j}$ is based on testing predicates $q_{p_{c_j}}$, $q = 1, \dots, n_p$ which take as arguments only the components of $c_{V_j}^i$. In pseudo-code it can be expressed as:

$$\text{if } q_{p_{c_j}}(c_{V_j}^i) \text{ then } c_j^{i+1} := m f_{c_j}(c_j^i) \text{ endif} \quad (12)$$

If in each step i only one out of the n_p predicates $q_{p_{c_j}}$ is true, then a single function $m f_{c_j}$ is selected. Usually in such a case $n_p = n_f$ and therefore the control program contains n_p instructions of the type (12) executed in an endless loop.

The system that is described in (Zieliński, 1994; Zieliński, 1995b) performs the selection of the function $m f_{c_j}$ for several consecutive steps, i.e. the selection is less frequent than the evaluation of the function, so the reaction is composed of several steps. In that case a behaviour is defined as a sequence of states:

$$b_j^i = \{c_j^{i+1}, c_j^{i+2}, \dots, c_j^{i+n_s}\} \quad (13)$$

where n_s is the number of steps in a behaviour (reaction). As this behaviour originates in c_j^i it is labelled with the superscript i , in other words it depends on c_j^i . The pseudo-code (12) represents a single-step behaviour, i.e. $n_s = 1$. In the case of a multi-step behaviour the pseudo-code assumes the following form:

$$\text{if } q_{p_{c_j}}(c_{V_j}^i) \text{ then } b_j^i(c_j^i) \text{ endif} \quad (14)$$

In the case (14) the decision as to which behaviour should be executed is taken once every n_s steps. Nevertheless, the transition between the state $c_j^{i+\epsilon}$ and $c_j^{i+\epsilon+1}$, where $\epsilon = 0, \dots, n_s - 1$ is still computed on the basis of the functions $m f_{c_j}$. The control program is composed of a loop containing a sequence of instructions of the form (14). If none of the predicates $q_{p_{c_j}}(c_{V_j}^i)$ is true a default behaviour, called the main reaction or a goal pursuing reaction, is executed. The other reactions deal with some abnormal situations – hindering attaining of the goal.

Rodney Brooks in (Brooks, 1986; Brooks, 1991) uses inhibition and suppression (exchange) mechanism for the evaluation of the final value of c_j^{i+1} – in this case the state of augmented finite state automaton. Each of the automaton generates signals that are the result of a process equivalent to the computation of functions $m_{f_{c_j}}$. The suppression and inhibition mechanism is equivalent to a selection based not only on the values of input signals but also on the priorities of automaton, i.e. priorities of the partial functions. In this way higher level behaviours can subsume the lower level ones.

Sometimes several predicates ${}^q p_{c_j}$ can be true simultaneously. In that case the values of several partial functions $m_{f_{c_j}}$ have to be composed together. Many composition operators can be conceived. Competitive methods are based on some form of selecting one value out of the computed values, e.g.:

$$c_j^{i+1} = \max_m m_{f_{c_j}}(c_j^i) \quad (15)$$

Cooperative methods are based on some form of superposition of the computed values, e.g.:

$$c_j^{i+1} = \sum_{m=1}^{n_f} m_{f_{c_j}}(c_j^i) \quad (16)$$

Weighted sums are the generalization of (16) and can be utilised also.

All sensor readings are intrinsically inaccurate due to signal noise, resolution etc. Moreover frequently a binary decision is impossible. An interesting possibility is the utilisation of fuzzy reasoning (Driankov *et al.*, 1993; Jantzen, 1999). Let the current sensor readings $c_{V_j}^i$ contain $\hat{c}_{V_j}^i$ – some of the readings that are of interest to us. Moreover, let $\hat{c}_{V_j}^i$ belong to a sensor reading subspace $\hat{C}_{V_j}$ – a part of the sensor reading space C_{V_j} . Each dimension of the space $\hat{C}_{V_j}$ forms a universe ${}_u \hat{C}_{V_j}$, $u = 1, \dots, n_u$, where n_u is the number of those dimensions (universes). Each universe ${}_u \hat{C}_{V_j}$ can be further subdivided into N_u regions ${}_h \hat{C}_{V_j}$, $h = 1, \dots, N_u$. Each of those regions forms a support for a fuzzy set F_u^h . To create a fuzzy set F_u^h an adequate membership function μ_u^h must be provided. The functions μ_u^h map ${}_u \hat{C}_{V_j}$ onto $[0, 1]$. For the sake of brevity the fuzzy sets and the values of linguistic variables representing them are not distinguished here.

Moreover, let either $c_{c_j}^{i+1}$ or $c_{e_j}^{i+1}$ contain the so called output value o_j^{i+1} . Beside that let that value belong to a space O_j . Here a one dimensional space is assumed. The space O_j will form an output universe. It must be divided into regions O_j^g , $g = 1, \dots, n_o$, which will form the supports for the output fuzzy sets ${}^o F^g$, where n_o is the

number of those regions and thus the number of the output fuzzy sets. Again, to create the fuzzy sets ${}^o F^g$ adequate membership functions ${}^o \mu^g$ must be provided. They map O_j onto $[0, 1]$.

Now the rule base can be constructed out of conditions formed by using the *AND* connective between the n_u fuzzy predicates (again linguistic variables are not used):

$$\begin{aligned} &\text{if } ({}_1 \hat{c}_{V_j}^i \text{ is } F_u^{\kappa_1}) \text{ AND } ({}_2 \hat{c}_{V_j}^i \text{ is } F_u^{\kappa_2}) \\ &\quad \text{AND} \dots \text{AND } ({}_u \hat{c}_{V_j}^i \text{ is } F_u^{\kappa_{n_u}}) \quad (17) \\ &\text{then } (o_j^{i+1} \text{ is } {}^o F^g) \end{aligned}$$

where: $\kappa_1 = 1, \dots, N_1$; $\kappa_2 = 1, \dots, N_2$; $\dots$; $\kappa_{n_u} = 1, \dots, N_{n_u}$. The rule base consists of N_r rules of the form (17). The fuzzification process converts each crisp value ${}_u \hat{c}_{V_j}^i$ into a degree of membership in F_u^{κ} . As a result we obtain:

$$\alpha_u^{\kappa} = \mu_u^{\kappa}({}_u \hat{c}_{V_j}^i) \quad (18)$$

where $\xi = 1, \dots, n_u$. Aggregation consists in computation of the firing strength of the condition of each rule. In the case of the *AND* connective used in (17) the firing strength of the rule ζ will be, e.g.:

$$\zeta_\alpha = \min_{u=1}^{n_u} \{\alpha_u^{\kappa}\}, \quad \zeta = 1, \dots, N_r \quad (19)$$

where in each case the adequate κ_ξ for the rule ζ are used. The firing strengths ζ_α are used to scale the output membership functions ${}^o \mu^g$. Those scaled membership functions form the conclusions (fuzzy sets) ${}^\zeta \beta$. The resulting fuzzy set is obtained through accumulation, e.g. performing a max operation on a member by member basis on the fuzzy sets ${}^\zeta \beta$, e.g.

$$\beta = \{\max_{\zeta=1}^{N_r} {}^\zeta \beta(o_j^{i+1})\} \quad (20)$$

The defuzzified crisp value of o_j^{i+1} is finally obtained by e.g. finding the abscissa of the centre of gravity of the set β . The thus computed value o_j^{i+1} is inserted into either $c_{c_j}^{i+1}$ or $c_{e_j}^{i+1}$. Multi-level fuzzy integration of behaviours is also possible, e.g. (Seraji and Howard, 2002).

4.2 Generalized hybrid deliberative-reactive systems

The purely reactive approach can be generalized if the predicates ${}^q p_{c_j}$ take as arguments not only $c_{V_j}^i$, but also other components of c_j^i , especially $c_{c_j}^i$. A purely deliberative system results if for the generation of c_j^{i+1} functions having as arguments only $c_{c_j}^i$ are used. This is rarely feasible in the case of embodied agents, as the state of the environment cannot be fully predicted, hence sensor

readings $c_{V_j}^i$ are necessary. In this way a hybrid deliberative-reactive systems result. Deliberation assumes the use of artificial intelligence techniques (Luger and Stubblefield, 1989) to find a plan (i.e. sequence) of actions leading the execution of a task (goal) set forth before the agent. In the case of deliberative or hybrid systems c_{e_j} must contain the representation of the state of the problem solution (not to be mistaken with the state of the environment or the agent itself). Moreover, within the agent there must exist operators, i.e. transition functions transforming one problem solution state into another. Those operators may result either from production rules or be the side-effect of application of predicate logic (Luger and Stubblefield, 1989). Deliberation is a search process starting in the initial state of the problem solution. This state includes a partial description of the current state of the agent, but also other search related information. As the operators are applied new problem solution states are generated. Heuristics help in discarding the produced states that either do not lead to a solution or are along a far from optimal search path, thus avoiding a combinatorial explosion in the search process. The path leading from the initial state and ending in a goal state describes a plan of actions that the agent should try to execute. Assuming that the plan generation starts with $c_{e_j}^i$, the plan (result of the search process) would be included in $c_{e_j}^{i+1}$ and would influence the generation of the state of the agent in the next steps, i.e. $c_j^{i+1}, c_j^{i+2}, \dots$. In the case of hybrid systems this plan can be treated as a goal pursuing reaction (main reaction). If the planning process takes a lot of time, the plan might not be ready in the instant $i + 1$. In that case the function (7) would have to generate c_j^{i+1} without the plan, so that c_j^{i+1} would have to result in halting the effector – to be on the safe side.

In the case of hybrid systems two approaches to plan generation can be followed. In the first case, the plan can be treated as a template and fixed corrective reactions handle minor problems with its execution. The plan constitutes a goal pursuing reaction and always the same corrective reactions are executed regardless of the step (action) in the plan. In the second case, the plan is generated with a changing set of corrective reaction for each action (plan step). This is equivalent to a system that changes both the goal pursuing reaction and the set of corrective reactions in each plan step. Nevertheless, in both cases such problems can arise that further execution of the plan will be impossible. In that case replanning has to be done – obviously with the current state of the agent included in the initial problem solution state.

Besides using $c_{e_j}^i$ as an argument of functions generating the value of c_j^{i+1} , also $c_{T_j}^i$ can be used. This implies direct cooperation of agents. Thus obtained information can be used both in the planning process as well as a parameter of the reactions.

5. IMPLEMENTATION ISSUES

The presented mathematical generic specification of multi-agent systems is based on transition functions and selection predicates. Both of those concepts are the basis of high-level general purpose computer programming languages. Mathematical functions can easily be transformed into a programming language (e.g. C, C++, Pascal) functions, procedures or OOP methods, while selection predicates are well grounded in control flow instructions of programming languages (e.g. in C: if then else, switch etc.). Thus the transformation of the specification into software is straight forward. Specific cases of such transformations have been presented in (Zieliński, 1994; Zieliński, 2001).

Usually the control subsystem of an agent is not implemented as a single entity (e.g. process). In that case there is no direct link between e_j and c_{e_j} . Both the input and the output effector images are being transformed through several intermediate layers. For example, in MRROC++ the Effector Control Process (ECP), which is equivalent to the agent's control subsystem, sends motion commands formed in the output effector image and receives the updates of the input effector image through Effector Driver Process (EDP) and Servo Group Process (SG) (Zieliński, 1999; Zieliński *et al.*, 2001). This only introduces multi-stage transmissions: $c_{e_j}^{i+1} \rightarrow e_j^{i+1}$ and $e_j^i \rightarrow c_{e_j}^i$, i.e.:

$$\begin{aligned} c_{e_j}^{i+1} &\rightarrow c_{e_j'}^{i+1} \rightarrow c_{e_j''}^{i+1} \rightarrow e_j^{i+1} \\ e_j^i &\rightarrow c_{e_j'}^i \rightarrow c_{e_j''}^i \rightarrow c_{e_j}^i \end{aligned} \quad (21)$$

Sometimes the creation of a virtual sensor reading require a multi-stage process, e.g. Arkin (Arkin, 1998) defines perceptual schemas that have that property. This also can be dealt with by a multi-stage aggregation and transmission similar to the one presented by (21).

An agent might need to change its immediate goal upon changes in the environment or the state of the other agents. One of the reasons for this change might be the replanning process mentioned earlier. To encode this type of behaviour it is convenient to use FSA (Finite State Automaton/Acceptors) (Arkin, 1998). In this case within c_{e_j} a transition graph (usually represented as a transition table) would have to be encoded. With each node of the graph a behaviour or a

set of concurrently executed behaviours would be associated. Upon termination of those behaviours (caused either by attaining the goal or permanent inability of reaching that state) the selection of the next graph node is made based on the current state of sensors $c_{V_j}^i$, state of the other agents $c_{T_j}^i$, and the agent's own current state $c_{c_j}^i$. This is performed by a transition function, so this idea is well grounded in the presented formalism.

An important aspect of an agent's activity is learning. For that purpose its control subsystem or a hierarchically higher entity must contain a critique. Its responsibility is to decide the result of the agent's actions. If the result is satisfactory either nothing is done or further tuning might be performed in hope of further improvement. If the result is unsatisfactory tuning must be done. Two kinds of tuning can be done. Either parameters of reactions are tuned (reinforced or weakened) or the whole code of reactions is changed. The second version is possible only if the code of the reactions is interpreted and not in binary form. In the first case the parameters of the functions (6) are modified and in the second case whole functions are altered.

6. CONCLUSIONS

Transition function based formalism facilitates the design and implementation of multi agent systems by decomposing a large system into components that can be designed and implemented by providing the code for the specified functions. Each of the control subsystem components (8) can be treated as an object in an object-oriented programming sense. Thus the communication with the other agents, effectors or virtual sensors can be handled internally by the methods of these objects (MRROC++ uses this method). The objects provide through their public interfaces only the data that is necessary for the computation of the control of the agent. Those objects provide data that is utilised by transition functions (6) resident in the control subsystem to compute the next state of this subsystem.

The formalism by enumerating the arguments of the transition functions (6) ensures that all the necessary interconnections between the components of the system will be present. The presented examples showed the numerous possibilities of designing such systems, e.g. purely reactive agents with diverse methods of composition of the final control signals, deliberative systems and hybrid deliberative-reactive systems. Nevertheless, the presented formalism also poses many open research problems, especially regarding detailed methods of controlling an agent. For instance: for a given task the selection of behaviours b_j^i

of an agent a_j is not obvious. Even less obvious is the assignment of subtasks to agents in a multi-robot behavioural system. The definition of functions ${}^m f_{c_j}$ defining a behaviour and the selection of predicates ${}^q p_{c_j}$ triggering a behaviour is also a difficult job. The treatment of abnormal situations (hardware or software errors) in both behavioural and deliberative systems also requires research. However, it should be noted that the enumerated problems pertain to the implementation of a specific activity of an agent and not to the structure of the system, hence the presented formalism serves its purpose of providing a formal representation for describing the structure of multi-agent programming frameworks. It points out what has to be contained in the framework both in terms of building blocks and in terms of its structure, i.e. what are the legal ways of assembling those building blocks. The formalism can also be used for describing the functioning of a single agent, but the control algorithm is external to the introduced notation. In other words, first we must have an idea of the control algorithm and only later we can express it in the presented formalism. The formalism only is a specification tool that facilitates the later implementation of the control system.

REFERENCES

- Alami, R., R. Chatila, S. Fleury, M. Ghallab M. and Ingrand F. (1998). An architecture for autonomy. *Int. J. of Robotics Research* 17(4), 315-337.
- Ambler, A. P. and D. F. Corner (1984). RAPT1 user's manual. Department of Artificial Intelligence, University of Edinburgh.
- Arkin, R. C. (1998). *Behavior-Based Robotics*. MIT Press. Cambridge, Mass.
- Backes, P., S. Hayati, V. Hayward and K. Tso (1989). The KALI multi-arm robot programming and control environment. In: *Proc. NASA Conf. on Space Telerobotics*.
- Blume, C. and W. Jakob (1984). Design of the structured robot language (SRL). In: *Advanced Software in Robotics* (A. Danthiène and M. Gérardin, Eds.). pp. 127-143. Elsevier. North Holland.
- Blume, C. and W. Jakob (1985). *PASRO: Pascal for Robots*. Springer-Verlag. Berlin.
- Blume, C. and W. Jakob (1986). *Programming Languages for Industrial Robots*. Springer-Verlag. Berlin.
- Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation* RA-2(1), 14-23.
- Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence* (47), 139-159.

- Driankov, D., H. Hellendoorn and M. Reinfrank (1993). *An Introduction to Fuzzy Control*. Springer-Verlag. Berlin, Heidelberg.
- Fleury, S. and M. Herrb (2001). Genom user's guide. Report, LAAS, Toulouse.
- Hayward, V. and R. P. Paul (1986). Robot manipulator control under unix RCCL: A robot control C library. *Int. J. Robotics Research* 5(4), 94–111.
- Hayward, V. and S. Hayati (1988). KALI: An environment for the programming and control of cooperative manipulators. In: *Proc. American Control Conference*. pp. 473–478.
- Hayward, V., L. Daneshmend and S. Hayati (1989). An overview of KALI: A system to program and control cooperative manipulators. In: *Advanced Robotics* (K. Waldron, Ed.), pp. 547–558. Springer-Verlag. Berlin.
- Jantzen, J. (1999). Design of fuzzy controllers. Technical University of Denmark, Department of Automation.
- Lloyd, J. and V. Hayward (1993). Real-time trajectory generation in Multi-RCCL. *J. of Robotics Systems* 10(3), 369–390.
- Luger, G. F. and W. A. Stubblefield (1989). *Artificial Intelligence and the Design of Expert Systems*. Benjamin-Cummings. Redwood.
- Markiewicz, M. E. and C. J. P. Lucena (2001). Object oriented framework development. *ACM Crossroads*.
Also: <http://www.acm.org/crossroads/xrds7-4/frameworks.html>.
- Morales, E. R. (1999). Generis: The ec-jrc generalised software control system for industrial robots. *Industrial Robot* 26(1), 26–32.
- Mujtaba, S. and R. Goldman (1979). AL users' manual. Stanford Artificial Intelligence Lab.
- Paul, R. (1977). WAVE – a model based language for manipulator control. *The Industrial Robot* pp. 10–17.
- Petersson, L., D. Austin and H. Christensen (2001). Dca: A distributed control architecture for robotics. In: *Proc. Int. Conference on Intelligent Robots and Systems IROS'01*.
- Russell, S. and P. Norvig (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall. Upper Saddle River, N.J.
- Schlegel, C. and R. Wörz (1999). Interfacing different layers of a multilayer architecture for sensorimotor systems using the object-oriented framework smartsoft. In: *3rd European Workshop on Advanced Mobile Robots, Eurobot'99, Zürich, Switzerland*.
- Seraji, H. and A. Howard (2002). Behaviour-based robot navigation on challenging terrain: A fuzzy logic approach. *IEEE Transactions on Robotics and Automation* 18(3), 308–321.
- Simmons, R. and D. Apfelbaum (1998). A task description language for robot control. In: *International Conference on Intelligent Robots and Systems IROS'98, Victoria, Canada*.
- Simmons, R., R. Goodwin, C. Fedor J. and B. B. B. (1997). Task control architecture: Programmer's guide to version 8.0. Carnegie Mellon University, School of Computer Science, Robotics Institute.
- Taylor, R. H., P. D. Summers and J. M. Meyer (1982). AML: A manufacturing language. *International Journal of Robotics Research* 1(3), 842–856.
- User's Guide to VAL II: Programming Manual Ver.2.0* (1986). Unimation Incorporated, A Westinghouse Company.
- Zieliński, C. (1991). TORBOL: An object level robot programming language. *Mechatronics* 1(4), 469–485.
- Zieliński, C. (1992). Object level robot programming languages. In: *Robotics Research and Applications* (A. Morecki et.al., Ed.). pp. 221–235. Warsaw.
- Zieliński, C. (1993). Flexible controller for robots equipped with sensors. In: *9th Symp. Theory and Practice of Robots and Manipulators, Ro.Man.Sy'92, Udine, Italy, Lect. Notes: Control and Information Sciences* 187. pp. 205–214. Springer-Verlag. Berlin.
- Zieliński, C. (1994). Reaction based robot control. *Mechatronics* 4(8), 843–86.
- Zieliński, C. (1995a). Control of a multi-robot system. In: *2nd Int. Symp. Methods and Models in Automation and Robotics MMAR'95, Międzyzdroje, Poland*. pp. 603–608.
- Zieliński, C. (1995b). *Robot Programming Methods*. Publishing House of Warsaw University of Technology. Warsaw.
Also: <http://www.ia.pw.edu.pl/~zielinski/>.
- Zieliński, C. (1997). Object-oriented programming of multi-robot systems. In: *Proc. 4th Int. Symp. Methods and Models in Automation and Robotics MMAR'97, Międzyzdroje, Poland*. pp. 1121–1126.
- Zieliński, C. (1999). The MRROC++ system. In: *1st Workshop on Robot Motion and Control, RoMoCo'99, Kiekrz, Poland*. pp. 147–152.
- Zieliński, C. (2001). By how much should a general purpose programming language be extended to become a multi-robot system programming language?. *Advanced Robotics* 15(1), 71–95.
- Zieliński, C., A. Rydzewski and W. Szykiewicz (1998). Multi-robot system controllers. In: *Proc. of the 5th International Symposium on Methods and Models in Automation and Robotics MMAR'98, Międzyzdroje, Poland*. Vol. 3. pp. 795–800.
- Zieliński, C., W. Szykiewicz, K. Mianowski and K. Nazarczuk (2001). Mechatronic design of open-structure multi-robot controllers. *Mechatronics* 11(8), 987–1000.